REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 12-22-95	3. REPORT TYPE AND DATES COVERED Final Report, 7/1/91 through 12/31/94

4. TITLE AND SUBTITLE
Synthesis of Timing-Constrained VLSI Systems

5. FUNDING NUMBERS
N00014-91-J-4041

6. AUTHOR(S)
Gaetano Borriello
Steven M. Burns
Carl Ebeling
Larry Snyder

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Washington
Seattle, Washington 98195

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Our research investigated the problem of synthesizing timing-constrained systems, with an emphasis on real-time control circuits and communication-intensive systems. Solving the general problem of synthesizing timing-constrained systems requires solutions to subproblems along a broad front from high-level specification to circuit design and implementation. The specific subproblems we investigated were 1) timing specification, analysis and verification, 2) high-performance clocking methodologies, 3) synthesis of reactive embedded systems, and 4) FPGA architectures and design tools for high-performance circuits and interfaces.

14. SUBJECT TERMS

19960122 101

15. NUMBER OF PAGES
26

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Northwest Laboratory for Integrated Systems
Department of Computer Science and Engineering
University of Washington

Synthesis of Timing-Constrained VLSI Systems

Final Report

November 28, 1995

Reporting Period:	July 1, 1991 — October 14, 1994
Principal Investigators:	Gaetano Borriello Steven M. Burns Carl Ebeling Lawrence Snyder
Sponsored by:	Advanced Research Projects Agency - CSTO Issued by Office of Naval Research Under Contract #N00014-91-J-4041
Technical contact:	Gaetano Borriello, 206/685-9432, fax: 206/543-2969 [gaetano@cs.washington.edu]
Administrative contact:	Sandi Larsen, 206/685-3796, fax: 206/543-2969 [sml@cs.washington.edu]

Table of Contents

1 Problem Statement

Our research investigated the problem of synthesizing timing-constrained systems, with an emphasis on real-time control circuits and communication-intensive systems. Timing-constrained systems are not necessarily high-performance systems, although they usually rely on high-performance implementation techniques. Timing-constrained systems are those that have specific constraints on their input-output behavior. Real-time systems, embedded systems, and communication systems generally have strict I/O timing constraints in addition to performance requirements. Solving the general problem of synthesizing timing-constrained systems requires solutions along a broad front from high-level specification to circuit design and implementation:

- **Timing specification, analysis and verification.** In order to synthesize timing-constrained systems, there must be a way to augment the functional specfication with a specfication of the timing constraints. The actual implementation of a system must be verified with respect to the timing constraints. This implementation may be partial or complete and the results of the analysis may be fed back to the synthesis algorithm to allow iterative improvement of the implementation.
- **High-performance clocking methodologies.** With increasing clock rates in high-performance systems, better clocking methodologies using level-sensitive latches and dynamic circuits need to be used. The increased complexity of this clocking methodology met be handled by automatic synthesis tools.
- **Synthesis of reactive embedded systems.** Synthesis tools for co-design have done little to address the role of timing constraints in portability and system integration. We seek ways to be able to specify the timing constraints on interactions with the processors' environment (devices and communication interfaces) so that it can be used to automatically synthesize interface glue logic and low-level device drivers. Thus, making designs easier to port and faster to integrate and prototype.
- **FPGA architectures and design tools for high-performance circuits and interfaces.** Field-programmable gate arrays (FPGAs) have become increasingly important for implementing "glue logic" and interfaces because they can be quickly and easily reconfigured to implement changing or enhanced protocols. Unfortunately, current commercial FPGAs do not support timing-predictable circuit implementation and moreover have no support for asynchronous circuits for implementing interfaces.

Our research attacked each of these subproblems, generating new specification techniques, algorithms, design tools and architectures. In addition, several implementation projects were done to gain experience with synthesizing timing-constrained systems.

2 Summary of Research Results

2.1 Timing Specification, Analysis and Verification

We have developed efficient algorithms for various timing analysis problems that have application to the synthesis and verification of digital systems. We focused on timing problems that arise in the synthesis of high-performance synchronous systems using either edge-triggered registers or level-sensitive latches as primitive storage elements. We have also developed practical algorithms for the verification and iterative re-synthesis of interface logic based on specifications of timing diagrams. We have also developed practical algorithms for the timing analysis of clockless (asynchronous) systems. These algorithms can be applied to the verification and iterative re-synthesis of not only pure asynchronous systems but also mixed synchronous/asynchronous systems.

2.1.1 Interface Specification, Synthesis, and Verification

Elizabeth Walkup, Gaetano Borriello

We conducted research in the area of interface specification in two areas. The first seeks to develop practical algorithms that can be used for verification and synthesis of interface logic. The second is to use interface specifications to automatically synthesize software device drivers and any accompanying interface hardware for the processor. One of our results in the first area also solves an important open problem in Discrete Event Systems.

Interface specifications are most often expressed in terms of timing diagrams. In building on our previous work in formalizing timing diagrams we investigated mapping timing diagrams to signal transition graphs. New techniques in STG synthesis for both synchronous and asynchronous methodologies make this a viable path for interface logic synthesis. In a paper that appeared in DAC'94, we presented a taxonomy for interface timing constraints and distinguished the synthesis and verification problems. In verification, we are interested in whether there is a possible assignment of circuit delays that will permit all the constraints to be satisfied. The results cover a range of possible implementations. On the other hand, in synthesis, we are interested in how much time is available to implement a particular logic path, taking into account the relation to other paths. The difference between the two problems is basically one of independent and dependent delay values, that is, where the choice of implementation in one part of the circuit affects the choices that can be made in another. We have developed an efficient algorithm for the synthesis problem that has two improvements over existing approaches: it has a much improved expected running time and properly handles delay bounds that are infinite (both necessary requirements if it is to be used to determine delay constraints on logic yet to be synthesized). The algorithm has been implemented and demonstrated to be highly effective on a wide range of sample problems. Recently, we have shown that this timing verification algorithm is the basis for a technique for solving general linear equations over the $(\max, +)$ algebra. This algebra, in which the max operation replaces addition, and $+$ replaces multiplication, has been developed in the Discrete Event Systems community and can be used to describe a wide range of problems in domains including planning, scheduling, and production.

Interface synthesis is also a problem in hardware/software systems. Given a collection of devices that must be controlled by a microprocessor or microcontroller we must generate the requisite interface hardware as well as the software device drivers that allow the application or control program to interact with the devices. We built on our previous work in port allocation in microcontroller-based systems. In a paper ICCAD'92 we presented an approach that minimizes the external hardware needed to connect the devices to the controller and then customizes skeletal software drivers to correspond to the synthesized interface hardware. In an article that appeared in the August 1994 issue of IEEE Micro magazine, we presented an algorithm based on max-flow min-cut methods that is used to partition interface events into hardware and software implementation based on edge weights derived from timing and physical constraints. This capability will permit us to automatically resynthesize tuned device drivers as microprocessors change in an implementation.

2.1.2 Time Separation of Events

Steve Burns, Henrik Hulgaard, Tod Amon, Gaetano Borriello

Determining the time separation of events (TSE) is a fundamental problem in the analysis, synthesis, and optimization of concurrent systems. Applications range from logic optimization of asynchronous digital circuits to evaluation of execution times of programs for real-time systems. We have developed an efficient algorithm to find exact (tight) bounds on the time separation of events in an arbitrary concurrent process specification without conditional behavior. We have recently extended our algorithm to consider specifications with conditional behavior.

The TSE problem occurs in two forms. The first problem is: given a description of a concurrent system as a set of events and rules with time intervals constraining the firing of these events, determine the largest possible separation in time between the firing of two particular event occurrences. The second problem is: determine this maximum separation over all occurrences of two events with the same difference in occurrence numbering. To explain further, given two events s and t and difference in occurrence index β, we want to find, for all possible firing times $\tau(s_{\alpha-\beta})$ and $\tau(t_\alpha)$ of the $(\alpha-\beta)^{\text{th}}$ and α^{th} occurrence of s and t, respectively, the tightest bound Δ such that

$$\tau(t_\alpha) - \tau(s_{\alpha-\beta}) \leq \Delta, \qquad \text{for all } \alpha \geq \max(0, \beta)$$

The major contribution of our work is a structural decomposition technique to solve the second problem. We implicitly analyze an infinite unfolding of our concurrent process specification and thus determine the tightest possible bound on Δ. This decomposition technique and its underlying algebraic formulation enable the algorithm to be efficient in practice. Furthermore, our algorithm handles arbitrary concurrent process specifications (without conditional behavior) and is thus useful in a variety of domains. In addition, we have generalized the algorithm to handle process specifications that are not necessarily strongly-connected, further increasing its range of applicability.

We have applied this algorithm to problems in a number of different application domains. These include: optimization of an asynchronous memory management unit, analysis of instruction execution times of an asynchronous microprocessor, analysis of a high-performance mixed asynchronous/synchronous communication interface, and isochronic fork analysis in asynchronous circuit synthesis.

In adapting our algorithm to include conditional behavior, we have concentrated on developing the analysis techniques necessary to aid in the synthesis of timed (no longer speed-independent) implementations of regularly structured circuits, e.g., stacks and queues. We can analyze the time separations for a particular type of stack, called an eager stack, that exhibits both conditional and concurrent behavior. For this parameterized circuit, the runtime of the analysis is polynomial in the size of the specification. Specifications with as many as 3000 nodes and 10^{16} reachable states can be readily analyzed. Existing systems explicitly enumerate the reachable states of the system and thus have inherently exponential runtimes.

2.2 High-performance clocking methodologies

2.2.1 Retiming of Level-Clocked Circuits

Soha Hassoun, Brian Lockyear, Carl Ebeling

We have developed an efficient algorithm for retiming circuits that use level-sensitive latches. This is a non-trivial extension of the original retiming work done by Leiserson and Saxe for circuits with edge-triggered registers. Level-sensitive latches are used in high-performance circuit designs because they are inherently faster and because more of the clock cycle can be devoted to computation. The timing analysis of such circuits, however, is very difficult because the latches allow values to propagate while the clock is high. Our work first defines what it means for a circuit to be correctly timed and then shows how to enumerate the constraints that must be satisfied by a correctly timed circuit. These timing constraints are of the form that can be solved easily using a fast shortest-path algorithm. Our algorithm works for circuits with any number of phases and any clock schedule that does not rely on minimum delays. This work was first published at the Brown/MIT VLSI Conference in March 1992 and subsequently in IEEE Transactions on Computer Aided Design. A second journal paper on extensions for asymmetric clock phases is under review.

We then extended these basic results by incorporating clock skew into the model. Clock skew is becoming an increasingly important problem as on-chip delays become relatively large compared to logic delays. Not only does clock skew reduce the performance of a circuit, it can also cause a circuit to fail for any clock speed. The first extension incorporated clock skew into the retiming problem. Two forms of skew are modeled: "fixed" skew, or that which is caused by or designed into the clock distribution network and known to the designer; and "variable" skew, that which is caused by fluctuations in process and operating conditions such as ambient temperature. In addition to including these parameters into our circuit and timing models, we add a parameter expressing "tolerance" to errors in clock skew estimates. By keeping the clock period constant and increasing the required tolerance, a designer can minimize a circuit's sensitivity to uncontrolled clock skew and errors in estimates of clock arrival time. This work was published in ICCAD-93.

The second extension combined retiming with "reskewing." Reskewing intentionally skews clock signals by placing delay elements into the clock distribution network. By adding these delays, the designer can shift the clock by a known amount and so increase the computation time available prior to latches driven by the delayed signals and decrease the time available after them. Thus the designer can change the "virtual" location of the latch without changing its physical location; placing the latch "inside" a combinational logic node at a location unreachable by retiming. Retiming and reskewing are complementary techniques, with retiming used to physically place synchronizers as close as possible to their optimal locations and reskewing to fine tune their local placement. We have developed an algorithm which combines retiming and reskewing for edge-clocked circuits. Our preliminary results on this problem appeared in Tau'93.

Although we have advanced the theoretical capabilities of retiming extensively, it remains a little used technique in practice. We believe that one reason for this is that it is generally thought of as a final optimization step, one to be applied late in the design process. By this point an ad hoc synchronizer placement and a variety of other timing optimization techniques have already been applied to the circuit design, leaving little room for further improvement by final stage retiming. We have described a methodology for using retiming earlier, as a part of the iterative development of a high-level design description. Retiming can be useful at this level in hiding the details of circuit timing and allowing synchronizers to be initially placed where they make clear logical sense instead of where they meet optimal timing goals. Retiming is then used to optimally place the synchronizers and to provide information about the critical paths in the resulting circuit. That critical path information can then be used to modify the high-level architecture appropriately.

Each of the algorithms developed as a part of this research effort have been implemented in a retiming tool at the University of Washington. This tool, SSkew, has been interfaced to the UW WireC design environment and the Berkeley SIS logic synthesis tools. This tool has been used to retime a wide range of benchmark circuits. The results of these experiments show that level-clocked retiming can substantially improve the performance of pipelined circuits. A description of this tool and the experimental results were published in 1994 in Brian Lockyear's Ph.D. thesis.

2.3 Synthesis of Reactive Embedded Systems

Our research goal is to develop a suite of tools to aid in the design of embedded systems, i.e., designs consisting of programmable components and custom logic and operating under hard real-time or performance constraints. Our target implementation media are off-the-shelf parts: primarily microprocessors and programmable logic arrays. The focus of our tools is on the retargetability of a single high-level specification to multiple implementation architectures. We seek to automatically partition and map the functionality desired onto the available components and provide information to the designer regarding system bottlenecks where the implementation architecture may be improved in terms of performance, cost, size, and/or power consumption.

The collection of tools make up the Chinook system. The first version can synthesize small single-processor designs producing the complete set of design data needed (netlist and code) to actually construct the system. The future versions will include static and dynamic scheduling of real-time code and support for partitioning software onto multiple processors.

We also engaged in several embedded system design projects that served to ground the research but are interesting in their own right. In addition they serve as examples for our rapid prototyping work and provide motivation for new system components such as processors with integrated FPGA-like interfaces.

2.3.1 Partitioning

Pai Chou, Ross Ortega, Henrik Hulgaard, Gaetano Borriello

In the synthesis of embedded systems we must consider partitioning the functionality to be implemented among multiple software engines and/or special-purpose hardware. Partitioning decisions are motivated primarily by real-time and performance constraints. In making these decisions, it is also important to consider how the elements will communicate once they are separated. Previous approaches to this problem have considered very fine-grain partitioning (at the level of program statements) and insisted on communication involving complex shared-memory or operating system support.

Our approach to this problem is to analyze the timing behavior of our specification to determine its schedulability on the selected processors. If there is no feasible schedule then we need to determine where to separate the specification onto two execution engines. To accomplish this, we require a powerful timing analysis algorithm (see section on timing analysis) that can answer questions posed by the scheduler. We focused on inter-process communication schemes that efficiently map to the processors used in typical embedded systems. In addition, we developed techniques for determining which portions of the software should be interrupt-driven and which should exploit statically-scheduled code. The method synthesizes parallel thread management code customized to the needs of the application and statically schedule code within the threads. The intra-thread code typically includes device driver functions that must satisfy real-time or performance constraints.

Another partitioning problem we consider is at the interfaces between processors and peripheral devices. Frequently, tight timing constraints on the protocols of the devices prohibit a processor from controlling the device directly. This is especially true for high-speed communication interfaces. In these situations, we need to synthesize external interface logic (including buffers and control finite state machines) to manage the interaction. Timing analysis and architectural-level retiming are instrumental in the task of modifying device drivers so that they control the device through the synthesized intervening hardware. We separate this partitioning problem from those at the inter-thread level as they operate under radically different constraint scales (nanoseconds vs. milliseconds). For more on this, see the subsection below on interface synthesis.

2.3.2 Software Synthesis

Pai Chou, Gaetano Borriello

There are many aspects to synthesizing real-time code. First among these is how to specify the timing constraints that the code must satisfy. We have developed a specification technique based on hierarchical and parallel *modes* that allow us to specify constraints unambiguously between I/O operations and include latency, throughput, and signalling constraints. In addition, the notation permits the specification of timeouts and interrupts as well as the conditions under which the constraints are in effect.

We have developed an efficient and general scheduling technique called *interval scheduling* for software written in this style. It takes as input a hierarchical mode-chart and bounds on the execution interval of each basic block of code (obtained from an estimating compiler) and produces serialized code that will meet all the constraints specified. This may involve unrolling of loops and interleaving of parallel modes. This is the first fully general scheduling method for real-time code that is guaranteed to find a solution if one exists and that can deal with bounds on the time of execution of operations rather than a fixed delay.

The timing analysis algorithms for concurrent systems outlined in section 2.1 is instrumental in guiding the partitioning decisions. By providing information on the separation between system events it enables an appropriate implementation that meets the constraints rather than an overly conservative one (as is frequently the case today) that wastes hardware resources and processor cycles.

2.3.3 Interface Synthesis

Elizabeth Walkup, Ross Ortega, Pai Chou, Gaetano Borriello

Interfaces in an embedded system exist at several levels. Processors must be connected to the device they control as well as to each other. This may be through shared communication media such as buses or point-to-point connections. Interfaces are composed not only of the glue logic necessary to make the logical and physical connection but also of the software that makes up the *device drivers*. As our focus was on retargetability, we developed methods for the high-level specification of device interfaces independent of the processor that will be driving them. Given a library we will provide tools for automatically customizing the device driver to the particular processor being used in a system. Designers writing a high-level specification of their system will be able to use an abstract procedural interface to peripheral devices and thereby improve the modularity of their specifications. One of the challenges is in propagating performance constraints of the application down into the device driver extracted from the library.

By taking the constraints on the device interface and the higher-level constraints from the application we can determine how to implement the interface to the device. This may include some external hardware to the processor to help with stringent interface timing or fast rate constraints. The hardware will typically include finite state machines and buffers. We now have an algorithm to determine this low-level partitioning automatically and generate both the customized device driver code as well as a specification for a logic synthesis tool to generate the glue logic. Previous work has determined how to connect multiple device interfaces to the same processor while minimizing multiplexing and decoding logic.

Note that inter-processor communication also falls into this model of devices and device drivers that must be customized. Techniques developed for device driver synthesis can be applied to the synthesis of communication channels between partitioned software running on multiple processors. The communication paths between them can be viewed as synthesized devices which will require their own specialized drivers. The only difference is that these *inter-processor communication devices* will have many more high-level constraints.

2.3.4 Device Driver Synthesis

Elizabeth Walkup, Ross Ortega, Gaetano Borriello

Libraries of device models are central to any co-synthesis system. Designers writing a high-level specification of their system must be able to use an abstract procedural interface to peripheral devices for reasons of modularity and hiding of details. The co-synthesis system must then be able to find a generic description of the device in a library and customize it for the particular processor being used.

There are two important parts to this problem. First, we investigated techniques for specifying the interfaces of devices as a collection of driver routines to implement the basic atomic operations supported by the device. The specifications include details of signalling protocols and their timing constraints. The second part of the problem is customizing these specifications into device driver code to be executed on a processor. This customization is compounded by the fact that we may need to insert some intervening glue logic between the processor to adapt to the processor's external communication wires as well as satisfy timing constraints.

Thus, given a collection of devices that must be controlled by a microprocessor or microcontroller, we must generate the requisite interface hardware as well as the software device drivers that allow the application or control program to interact with the devices. In a paper that was presented at ICCAD'92 we presented an approach that minimizes the external hardware needed to connect the devices to the controller and then customizes skeletal software drivers to correspond to the synthesized interface hardware. We have also developed an algorithm based on max-flow min-cut methods that is used to partition interface events into hardware and software implementation based on edge weights derived from timing and physical constraints. This capability will permit us to automatically resynthesize tuned device drivers as microprocessors change in an implementation.

Techniques developed from device driver synthesis can also be applied in synthesizing communication channels between partitioned software running on multiple processors. The communication paths between them can be viewed as synthesized devices which will require their own specialized drivers.

2.4 FPGA Architectures and Design Tools for High performance Circuits and Interfaces

Our work with FPGA architectures and design tools is an ongoing research effort that began when the Triptych architecture was conceived. Triptych has been shown to have a factor of 4 higher density than current commercial architectures (in terms of effective gates per unit of silicon area). We have also extended the architecture to develop Montage, the first FPGA to directly support asynchronous circuits, thus providing a potentially new prototyping medium to a community that has relied on custom VLSI. In the process of developing these architectures and proving their effectiveness, we developed Emerald, a set of architecture exploration and mapping tools that focus upon obtaining both high density and high performance.

A closely related effort has been the development of architectures and tools to support Multi-FPGA systems. In this area we have proposed Springbok, a new architecture that is flexible enough to incorporate complex chips like memories and microprocessors while retaining FPGAs to model glue and interface logic. This effort has also spawned the development of new mapping tools to support such multi-FPGA systems, including new partitioning methods as well as pin assignment schemes that allow higher performance mappings.

2.4.1 Triptych: A New Field-Programmable Gate Array Architecture

Carl Ebeling, Gaetano Borriello, Scott Hauck, David Song, Elizabeth Walkup

Most general-purpose FPGAs use a combination of programmable logic blocks and programmable interconnect to provide a general circuit implementation structure. This clear separation between logic and interconnection resources is attractive because the mapping, placement and routing decisions are decoupled. The price for this separation is the large area and delay costs incurred for the flexible interconnection needed to support arbitrary routing requirements. This leads to architectures like Xilinx, where the routing resources consume more than 90% of the chip area.

Domain-specific FPGAs like the Algotronix CAL1024 and Concurrent Logic CFA6000 increase the chip area devoted to logic by providing a less general, nearest neighbor, routing structure appropriate for structured applications such as DSP and systolic algorithms. These FPGA architectures, however, are not suitable for general applications, particularly state machines and controllers.

We have designed a new FPGA architecture called Triptych which can efficiently implement both structured circuits like data paths and more general circuits like state machines. Triptych differs from other FPGAs by matching the structure of the logic array to that of the target circuits, rather than providing an array of logic cells embedded in a general routing structure. By matching the physical structure to the logical structure, we reduce the amount of random routing that is otherwise required. Triptych provides an underlying fanin/fanout tree structure that matches the general structure of multilevel logic DAGs. This basic structure is augmented with segmented routing channels between the columns that facilitate larger fanout structures than is possible in the basic array. Finally, two copies of the array, flowing in opposite directions, are overlaid. Connections between the planes exist at the crossover points of the short diagonal wires. It is clear that this array does not allow arbitrary point-to-point routing like that associated with Xilinx and Actel FPGAs. However, we claim that this array matches the form of a large class of circuits, and that a mapping strategy that takes this structure into account can produce efficient implementations.

We have measured the potential of the Triptych architecture relative to other reprogrammable FPGAs by manually mapping a range of interesting circuits, including structured circuits and random logic, and comparing the results with respect to area cost and circuit speed. This comparison shows that Triptych is similar in cost to the Algotronix CAL1024 for structured circuits suited to the CAL1024 and superior for circuits with non-local communication. Triptych is about twice as efficient as Xilinx across the whole range of circuits. The performance of Triptych implementations is comparable to that of other FPGAs.

2.4.2 MONTAGE: An FPGA for Synchronous and Asynchronous Circuits

Scott Hauck and Steve Burns

Field-programmable gate arrays provide an ideal implementation medium for system interface and glue logic. They integrate large amounts of random logic and simple data paths, and can be easily reprogrammed to reflect changes in system components. Unfortunately, most of the effort in designing FPGA architectures has ignored the special problems of these types of circuitry. Interface and glue logic require support for interfacing asynchronous logic to synchronous logic, interconnecting separately-clocked synchronous components, and controlling certain circuit delays, all of which are largely ignored by current architectures.

Asynchronous circuits are also not well served by current FPGAs. Implementations of asynchronous logic must consider hazards in the logic, synchronization and arbitration of events, and strict adherence to the timing assumptions of the design methodologies. Unfortunately, it is not possible to implement these circuits in a robust manner in current FPGAs. Some of the elements required (most importantly, arbiters that resolve conflicts between two concurrently arriving signals) are not implementable in the standard digital logic found in these devices. In addition, the logic and routing elements must be designed more carefully in order to avoid extra "glitches" on lines, since in asynchronous circuits every transition is important. Finally, routing resources must have predictable, optimizable delays in order to meet timing assumptions in the design methodologies.

However, the problems are not restricted simply to the architectures themselves. The mapping software also must be altered to handle the demands of asynchronous logic. Primarily, there are much stricter timing demands that must be upheld. Bundled data and isochronic forks both require signals to be routed with special delay demands, demands that impact placement of logic cells as well. Also, the logic decomposition used to break logic blocks down to the size required by an FPGA (the covering problem) cannot simply use the algorithms for synchronous logic. For quasi-delay insensitive circuits, where the only timing assumptions made are those of isochronic forks, standard synchronous logic decomposition techniques can add extra levels of logic incorrectly, putting hazards into the circuit.

In the Montage project we addressed these concerns both by developing a new FPGA architecture based on the Triptych architecture, but optimized for asynchronous and synchronous interface logic, and by writing mapping tools for this architecture. The Montage architecture includes support for arbiter and synchronizer elements via specialized cells intermingled with the standard Triptych-style blocks. Also, the feedback features of Montage have been enhanced to handle the added utilization from state-holding functions. Finally, the latch elements have been configured to work with several clocking schemes simultaneously active in the chip. This allows for two separately clocked synchronous systems to co-exist, as well as providing a novel initialization procedure for asynchronous stateholding elements.

2.4.3 Modeling of FPGA Architectures with Emerald

Darren Cronquist, Larry McMurchie

Designing a robust set of CAD tools for an industry-strength field-programmable-gate-array (FPGA) requires a great deal of dedication and resources. Fortunately, this development process need not begin from scratch since most phases in the FPGA tool path, such as technology mapping, placement, and routing, have an underlying structure that is common to all FPGA architectures. In fact, FPGA architectures themselves are similar enough to be characterized in a general way. As a result, a detailed FPGA description coupled with a rich set of architecture-adaptive tools can provide a powerful initial working environment for FPGA tool construction. These goals have been realized in the Emerald system.

The routing resources of an FPGA architecture consist of switches, multiplexers, demultiplexers and other programmable routing elements. Because of the topological nature of the resources, a schematic capture system is a natural starting point for specification. In Emerald we used WireC environment because of its ability to specify schematic information in a concise and flexible way.

Like other schematic editors, the WireC allows instances of devices and their connections to be specified graphically. WireC, however, extends the notion of a schematic editor to include procedural constructs. Consequently, elements can be conditionally and iteratively instantiated. This feature is particularly convenient for creating compact descriptions of structures which have a high degree of regularity, such as FPGAs. In addition, the parameterization of the schematics allows entire architecture families (differing in size and features) to be represented by the same drawings, which is convenient for creating scaled-down versions of an architecture for testing and debugging purposes.

2.4.4 A Performance-Driven Router

Carl Ebeling, Larry McMurchie

We have developed a router currently in use with Triptych/Montage FPGAs and applicable to a variety of FPGA architectures. Based on Nair's iterative global router,[1] this router uses the

[1]R. Nair, "A Simple Yet Effective Technique for Global Wiring", IEEE Transactions on CAD, vol. CAD-6 no. 2, 1987.

congestion found in previous iterations when routing signals in later iterations. We have introduced delay into this congestion-based scheme so as to maintain the performance of the mapped circuit.

The algorithm can be divided into two interacting parts which can be treated as operationally orthogonal. The first part, the signal router, routes signals given a cost and delay for each routing resource. The second part, the global router based on Nair's work, dynamically adjusts the cost of each resource based on the demand signals place on that resource. In this way signals may use a resource that is in high demand only if they are willing to pay the higher cost. This naturally forces signals to make use of resources that might not otherwise be used. The global router reroutes signals using the signal router until no resources are shared.

The signal router must balance two competing demands: low congestion cost and low delay. Low congestion cost can be achieved by a minimum spanning tree, but this can result in long delays. Low delay can be achieved by a minimum-delay tree, but this usually means a high congestion cost. The signal router uses information about the relative contribution of a signal to the overall delay of the circuit to determine how to balance these two concerns. After each iteration of the router, a timing analysis is performed which first finds the critical path (maximum delay), and thus the minimum clock cycle. A slack ratio is computed for each edge in the circuit, i.e. each source/sink pair, as the ratio of the delay of the most critical path using that edge to the maximum path delay (the critical path) in the circuit. Thus every edge on the critical path has a slack ratio of 1.0 while edges on the least critical paths have slack ratios close to 0.0.

The key idea behind the signal router is that edges with a large slack ratio should be routed directly from source to sink using a shortest path algorithm, while edges with a small slack ratio should be routed using a minimum spanning tree. The slack ratio allows a smooth tradeoff between low delay and low congestion.

This routing algorithm is currently being used as the router for the Triptych and Montage FPGAs. Results over a wide range of benchmark circuits indicate that the router rarely causes a slowdown of more than 5% in the circuit delay over the minimum delay route.

Using the Emerald system described elsewhere in this report, we have created an architecture description for the Xilinx 3000 family of FPGAs. Using the Xilinx placement tool (apr), we generated placements for a variety of circuits targeted to this architecture. Then we ran our router on these placements and evaluated the quality of the route using the Xilinx xdelay tool. The results show an improvement in the total routing delay by up to 23% over the Xilinx router.

2.4.5 CAD for Multi-FPGA Systems

Scott Hauck, Gaetano Borriello, Carl Ebeling

Multi-FPGA systems have shown a lot of promise recently. Systems from DEC Paris Research have yielded Supercomputer-class performance on numerous tasks, including a world-record speed for RSA encryption. Logic Emulation engines from Quickturn Inc. have delivered ASIC prototyping performance several orders of magnitude faster than standard software simulators or even hardware accelerators. Systems from GigaOps Inc., Virtual Computer Corp., as well as others, are even beginning to move these benefits to the microcomputer market.

We see two major problems in the state-of-the-art of multi-FPGA systems: A lack of effective CAD tools, and an inability to support the prototyping of complete systems. Our approach has been to develop a set of software mapping tools, as well as a new concept in flexible system prototyping to fill these needs.

Our software mapping system is based on the philosophies that the software must be completely automatic, and that it must be able to be applied to arbitrary (and flexible) multi-FPGA systems. We have investigated standard bipartitioning algorithms, integrating existing techniques with some novel algorithms. This has yielded an extremely fast partitioning algorithm that yields cut sizes 14% smaller than all other published work. We have also developed automatic means to apply this strategy to arbitrary multi-FPGA systems. This method determines how to best iteratively apply bipartition to maximize locality, thus minimizing the amount of inter-FPGA communication required. This is critical, because today's systems are greatly limited by their communication bandwidth.

We have also developed pin assignment software for arbitrary inter-FPGA topologies. To map to multiple-FPGA systems one needs to partition the logic into the various chips and then route the interchip signals. Unfortunately, one cannot handle the entire task of routing between FPGAs via standard FPGA or ASIC routers because the source and destination of the routes are not fixed. Specifically, although the logic has already been assigned to specific FPGAs, its location inside the FPGA is not yet fixed. This location will be determined via standard FPGA placement after the inter-chip signals have been routed, and the placement will depend on the inter-chip routes of all the signals connected to it. Our approach is to do routing in two phases. First, we decide which intermediate FPGAs to route through via standard routing algorithms. For this step each FPGA is a single node in the routing topology, and standard routing techniques can be applied. The second step is to assign specific pins for the signals to route through. This is done conceptually by using force-directed placement to place all of the FPGAs simultaneously. To make this tractable we have developed simplification rules. These are based on the fact that we are only concerned with the placement of I/O pins in the FPGAs, and thus the internal nodes can be removed. Springs that were connected to these nodes are replaced with equivalent springs connected between the node's neighbors. This pin assignment system is operational, and can find pin assignments for chips in a few minutes, while actual placement and routing via the vendor's commercial tools can take hours.

In terms of hardware for multi-FPGA systems, we have considered how to best construct multi-FPGA topologies. We have completed a quantitative study of mesh routing topologies in order to fix the basic structure of our substrate. These resulted in three complementary strategies for topologies that decrease inter-chip routing delays and resource utilization by more than 50% over simple 4-way Meshes. The strategies include both increasing the number of nearest-neighbors and changing the pattern with which signals from neighbors are placed at the input pins of the FPGAs.

Our work on system-level prototyping has led us to propose the Springbok architecture. This system is flexible enough to include complex chips from the user's design, which are either inefficiently handled by standard FPGA-only systems (such as microprocessors and memories), or cannot be handled at all (such as A/D converters and LCD screens). It is also expandable to handle larger systems and greater bandwidth requirements.

Related to the Springbok system is our effort on supporting real-world interfaces for FPGA-based prototyping systems. Existing solutions such as the Quickturn systems provide near-speed testing of ASICs, giving us hope that we can now test prototype devices in their target environment. However, the external interfaces of these chips are often required to support bandwidth and response times that are beyond the ability of today's FPGA-based prototypes. Our solution is to generate a system of FPGAs and memories to support these interfaces, handling both the bandwidth and response-time requirements. These are capable of filtering incoming data enough to allow the multi-FPGA system to keep up, while also speeding up the outgoing data to meet interface requirements. In this way, we get not only a flexible interface system for Springbok-based designs, but also a method equally applicable to interface requirements of current commercial systems.

2.5 Subgraph Isomorphism

Carl Ebeling

We have extended our previous algorithms for graph isomorphism used by the Gemini program to solve the subgraph isomorphism problem. Subgraph isomorphism is known to be NP-hard and all previous attempts to solve it in the context of circuits have resorted to a variety of ad hoc techniques. By contrast, SubGemini is based on general graph coloring techniques and thus applicable to a range of circuit technologies.

Subgraph isomorphism appears in many contexts. Examples include automatically identifying components in large VLSI circuits so that hierarchy can be extracted from layout, identifying possible coverings of logic graphs using library components and performing design-rule checks based on finding illegal circuit structures.

Our algorithm is based on graph coloring and operates in two phases. In the first phase, the subgraph is colored such that the color of each node is determined only by the internal structure of the subgraph. A node at the center of the subgraph is also identified as the keystone node. The target graph is also colored such that the keystone nodes of all subgraph instances (and possibly other nodes as well) have the same color as the subgraph keystone node.

In the second phase of the algorithm, each possible keystone node is examined in turn to verify that it is part of a subgraph instance. This is performed by coloring from the keystone node in both the subgraph and the target graph. Fortunately, it is relatively easy to show that nodes outside the subgraph instance can be kept from causing spurious colors on internal nodes. This coloring provides a match if one exists, but must rely in difficult cases on backtracking.

An implementation of this algorithm has been completed and results indicate that the algorithm works extremely well for practical circuits. This work was published in the Design Automation Conference in 1993.

2.6 The Chaos Router Chip

Kevin Bolding, Neil McKenzie, Larry Snyder, Carl Ebeling, Robert Wille, Soha Hassoun, Larry McMurchie

The Chaos router chip is a prototype of a two-dimensional (mesh and torus) chaotic router. The chip contains all of the functionality of a chaotic router and is designed to operate at speeds competitive with current state-of-the-art routers using similar fabrication technology.

The design process employed two models of the chip. The first is a transistor-level model produced directly from schematics. In these schematics, critical transistor sizes and capacitances were specified, but most instances were left at their default sizes. Simulations based on this model provided a relatively easy method of validating functional correctness and modifying the design as needed. The second model was generated using Cascade Design Automation's ChipCrafter tools. This model involved generating an actual layout of the chip based on the schematics. The chip layout was then extracted into a netlist using Mextra and simulated using IRSIM. With this model, reasonable capacitance estimates and actual transistor sizes were used so that performance estimates could be made.

The chip generated by ChipCrafter combined several different design styles. Random logic was specified in schematics and compiled into standard cell arrays by ChipCrafter. Major datapath components, such as the FIFO's and crossbars, as well as critical control components such as the multiqueue controller were designed using full custom logic. These components were tested using IRSIM and HSpice and integrated into the design using ChipCrafter. ChipCrafter performs automatic placement and routing at the chip level, but allows manual adjustments to the floorplan as well. We used this mode of ChipCrafter to generate layout for the entire chip.

By the end of fall 1992, all components were designed and integrated into ChipCrafter. Functional correctness was verifyed by early winter 1993. At this point, we began performance optimization in order to meet our goal of a 15 ns cycle time. By retiming the main routing pipeline and optimizing critical paths, we achieved this goal by the end of winter 1993. After careful floorplanning, we produced a layout for a die approximately 1 cm on a side with around 110,000 transistors. The chip contains five duplex (bi-directional) 16-bit channels, one for each of the North, South, East, and West directions, and one to connect the router to a processing node.

Functional testing was accomplished using the UW chip tester. The primary test exercised all of the control and datapath logic by using random test patterns over extended periods of time. This was accomplished by using the tester to emulate the channel protocols on each of the five channels as if the chip were part of a large network. Packets were introduced to each of the chips five channels whenever the channel was available. The testing software maintained a record of packets traveling on each of the channels and confirmed that packets were correctly routed.

The chip functioned flawlessly in oblivious mode, delivering all packets to their correct output channels in the correct order. In chaotic mode, almost all (more than 99%) of the packets are delivered correctly. A small logic bug exists that, when packets arrive on an input channel and are routed back to the same channel (loop back on themselves), sometimes creates extra packets which are a shortened copy of the original packet. This situation is easily detectable. Unfortunately, in very rare circumstances, the short packet may cause confusion to the control logic and result in failure of the entire router. In these cases, there is no easy way to recover other than resetting the router. The logic bug may be fixed easily with a re-fabrication of the chip. It was not found in pre-fabrication testing because it is rare and was not uncovered by the random test vectors simulated.

Because the UW tester cannot test the chip at its expected operational speed of 66MHz, a separate test apparatus was built to accomplish this. A set of four chaos chips will be connected together into a torus, with the UW tester connected to the processor port of one chip. The tester will be used to inject a series of packets into the network while the network is operating at a slow clock speed. The clock speed will then be raised to the test level, and the packets will circulate through the network until they reach their destinations. At this point, the clock can be slowed down again and the tester used to extract the packets from the network to confirm correct operation.

A prototype of this configuration has been built and the router tested to a speed of 22MHz. At this point, the limitations of the prototype hardware and the wire-wrap connections prevent faster speed testing until a PC-board test unit can be fabricated.

3 Bibliography

3.1 Patents

C. Ebeling, G. Borriello, S. Hauck, S. Burns, "Dynamically Reconfigurable Logic Array for Digital Logic Circuits," U. S. Patent #5,208,491, issued November 1992.

S. Hauck, G. Borriello, S. Burns, C. Ebeling. "A Field-Programmable Gate Array for Synchronous and Asynchronous Operation," U.S. Patent # 5,367,209, issued November 1994.

3.2 Journal Papers

P. Chou, E. Walkup, G. Borriello, "Scheduling Issues in Hardware/Software Co-Synthesis," *IEEE Micro* special issue on hardware/software co-design, Vol. 14 No. 4 pp. 37-47, August 1994.

S. Hauck, S. Burns, G. Borriello, C. Ebeling, "An FPGA For Implementing Asynchronous Circuits," *IEEE Design & Test of Computers*, Vol. 11, No. 3, pp. 60-69, Fall 1994.

B. Lockyear, C. Ebeling, "Optimal Retiming of Level-Clocked Circuits Using Symmetric Clock Schedules," *IEEE Trans. on CAD*, Vol. 13, pp. 1097-1109, September 1994.

S. Hauck, "Asynchronous Design Methodologies: An Overview," *Proceedings of the IEEE*, Vol. 83, No. 1, pp. 69-93, January 1995.

H. Hulgaard, S. Burns, T. Amon, G. Borriello, "An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems," *IEEE Transactions on Computers*, Vol. 44, No. 11, pp. 1306-1317, November 1995.

G. Borriello, C. Ebeling, S. Hauck, S. Burns, "The Triptych FPGA Architecture," *IEEE Transactions on VLSI Systems*, vol. 3, no. 4, pp. 491-501, December 1995.

S. Hauck, "Chapter 15: Design of Asynchronous Circuits," appearing in J. A. Brzozowski, C-J. H. Seger, *Asynchronous Networks*, Springer-Verlag, 1995.

C. Ebeling, L. McMurchie, S. Hauck, S. Burns, "Mapping Tools for the Triptych FPGA," *IEEE Transactions on VLSI Systems*, vol. 3, no. 4, pp. 473-482.

H. Hulgaard, S. Burns, G. Borriello, "Testing Asynchronous Circuits: A Survey," to appear in *Integration.*

N. Mckenzie, C. Ebeling, L. McMurchie, G. Borriello, "Experiences with the UW MacTester in Computer Science and Engineering Education," to appear in *IEEE Transactions on Education.*

B. Lockyear, C. Ebeling, "Optimal Retiming of Level-Clocked Circuits Using Asymmetric Clock Schedules," submitted to *IEEE Trans. on CAD.*

3.3 Conference Papers

C. Ebeling, G. Borriello, S. Hauck, D. Song, E. A. Walkup, "Triptych: A New FPGA Architecture", Oxford Workshop on Field-Programmable Logic and Applications, September 1991. Also appearing in W. Moore, W. Luk, Eds., FPGAs, Abingdon, England: Abingdon EE&CS Books, pp. 75-90, 1991.

S. Hauck, G. Borriello and C. Ebeling, "TRIPTYCH: An FPGA Architecture with Integrated Logic and Routing", *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pp. 26-43, March 1992.

P. Chou, R. Ortega, G. Borriello, "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems," *IEEE/ACM International Conference on Computer-Aided Design*, November 1992, pp. 488-495.

M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifying Subcircuits Using a Fast Subgraph Isomorphism Algorithm," *30th IEEE/ACM Design Automation Conference*, pp. 31-37, June 1993.

K. Bolding, S. Cheung, S. Choi, C. Ebeling, S. Hassoun, T. Ngo, R. Wille, "The Chaos Router Chip: Design and Implementation of an Adaptive Router," *VLSI '93*, September 1993.

H. Hulgaard, S. Burns, T. Amon, G. Borriello, "An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems," *International Conference on Computer Design*, pp. 166-173, October 1993, Awarded Best Paper in CAD Track.

H. Hulgaard, S. Burns, T. Amon, G. Borriello "Practical Applications of an Efficient Time Separations of Events Algorithm," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 146-151, November 1993.

B. Lockyear, C. Ebeling, "The Practical Application of Retiming to the Design of High-Performance Systems," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 288-295, November 1993.

S. Hauck, G. Borriello, S. Burns, C. Ebeling, "Montage: An FPGA for Synchronous and Asynchronous Circuits=B2", in H. Grunbacher, R. W. Hartenstein, Eds., Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping, Berlin: Springer-Verlag, pp. 44-51, 1993.

E. Walkup, G. Borriello, "Device Driver Synthesis," submitted to *1994 IEEE/ACM International Conference on Computer-Aided Design*, April 1994.

P. Chou, G. Borriello, "Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems," *31st ACM/IEEE Design Automation Conference*, pp. 1-4, June 1994.

E. Walkup, G. Borriello, "Interface Timing Verification with Application to Synthesis," *31st ACM/IEEE Design Automation Conference*, pp. 106-112, June 1994.

P. Chou, G. Borriello, "Interval Scheduling: Fine-grained Software Synthesis for Embedded Systems," submitted to *32nd ACM/IEEE Design Automation Conference*, pp. 462-467, October 1994.

S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems, " *ICCD '94*, pp. 170-177, October 1994.

H. Hulgaard, S. Burns, "Bounded Delay Timing Analysis of a Class of CSP Programs with Choice," *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 2-11, November 1994.

H. Hulgaard, T. Amon, S. Burns, G. Borriello, "Timing Analysis of Timed Event Graphs with Bounded Delays Using Algebraic Techniques," *Proceedings of the 33rd IEEE Conference on Decision and Control*, pp. 959-960, December 1994.

S. Hauck, G. Borriello, "Logic Partition Orderings for Multi-FPGA Systems," *FPGA '95*, Santa Cruz, CA, pp. 32-38, February 1995.

L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, pp. 111-117, February 1995.

3.4 Workshop Papers

S. Hauck, G. Borriello, S. Burns, C. Ebeling, "Montage: An FPGA for Synchronous and Asynchronous Circuits," *2nd IFIP Workshop on Field-Programmable Logic and Applications*, September 1992.

P. Chou, R. Ortega, G. Borriello, "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems," *IEEE International Workshop on Hardware/Software Co-Design*, October 1992.

G. Borriello, A. Sangiovanni-Vincentelli, "Models for the Hardware/Software Co-Design of Embedded Controllers," *IEEE International Workshop on Hardware/Software Co-Design*, October 1992.

S. Hassoun, G. Borriello, "Improving Finite State Assignment for Two-Level Programmable Logic Devices," *4th ACM International Workshop on Logic Synthesis*, poster, May 1993.

H. Hulgaard, S. Burns, T. Amon, G. Borriello, "Practical Applications of an Efficient Time Separation of Events Algorithm," *3rd ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.

B. Lockyear, C. Ebeling, "Minimizing the Effect of Clock Skew Via Circuit Retiming," *3rd ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.

E. Walkup, G. Borriello, "Interface Timing Verification with Combined Max and Linear Constraints," *3rd ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.

P. Chou, G. Borriello, "Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems,"

International Workshop on Hardware-Software Codesign, October 1993.

E. Walkup, G. Borriello, "Automatic Synthesis of Device Drivers for Hardware/Software Codesign," *International Workshop on Hardware-Software Codesign*, October 1993.

S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies For FPGA Arrays," *ACM International Workshop on Field Programmable Gate Arrays*, February 1994.

S. Hauck, G. Borriello, C. Ebeling," Springbok: A Rapid-Prototyping System for Board-Level Designs", *ACM/SIGDA 2nd International Workshop on Field-Programmable Gate Arrays*, February 1994.

S. Hauck, G. Borriello, "Pin Assignment for Multi-FPGA Systems," *IEEE Workshop on FPGAs for Custom Computing Machines*, April 1994.

G. Borriello, D. Miles, "Task Scheduling for Real-Time Multi-Processor Simulations," *11th IEEE Workshop on Real-Time Operating Systems and Software*, May 1994.

4 Scientific Personnel

Faculty

Gaetano Borriello
Steve Burns
Carl Ebeling
Lawrence Snyder

Research Staff

Larry McMurchie

Graduate Students

Tod Amon**
Kevin Bolding**
Sung-Eun Choi*
Pai Chou*
Darren Cronquist
Soha Hassoun*
Scott Hauck*
Henrik Hulgaard*
Jordan Lampe
Brian Lockyear**
Neil McKenzie*
Miles Ohlrich*
Ross Ortega*
Elizabeth Walkup*
Robert Wille*

* indicates that the student has received an M.S. degree.
** indicates that the student has received a Ph.D. degree.